Homework 9 Selected Solutions
APPM/MATH 4650 Fall '20 Numerical Analysis

Due date: Saturday, November 14, before midnight, via Gradescope. **Instructor**: Prof. Becker
Theme: ODEs and IVPs

solutions version 11/8/2020

Problem 1: Consider the following initial value problem

$$y''(t) + 5y'(t) + 6y(t) = \cos(t), \quad y(0) = 1,\ y'(0) = 0, \text{ for } t \in [0, 20] \tag{1}$$

Using the techniques you learned in your ODE class (APPM 2360 or MATH 3430), find an analytic solution. *Hint*: you are welcome to do the next problem and solve this numerically as a means to confirm that your derivation is correct.

Solution:

There are different techniques to solve this. The one I remember off the top-of-my-head is (1) find a homogenous solution, and then (2) find a particular solution by guessing cleverly chosen ansatz (since the forcing term is of a nice simple form).

(1) To find the homogenous solution, we solve $y''(t) + 5y'(t) + 6y(t) = 0$. The trick here is to try the ansatz $y(t) = e^{rt}$. Plug this in, and we get $r^2 y(t) + 5ry(t) + 6y(t) = 0$. Since $y(t) \neq 0$, we can divide by $y(t)$ and get $r^2 + 5r + 6 = 0$, which is a simple quadratic equation and we can either find the roots by observation $(r+3)(r+2) = r^2 + 5r + 6$ or via the quadratic formula. Since this is a homogenous solution to a linear equation, we have an entire subspace (of dimension 2) of solutions, so the general form is

$$y_h(t) = ae^{-3t} + be^{-2t}.$$

Since this is exponential decay, this represents the transient part of the solution.

(2) For the particular solution, guess $y_p(t) = A\cos(t) + B\sin(t)$, and plug this into the equation. Noting $y_p' = -A\sin(t) + B\cos(t)$ and $y_p''(t) = -A\cos(t) - B\sin(t)$, we solve

$$(-A\cos(t) - B\sin(t)) + 5\left(-A\sin(t) + B\cos(t)\right) + 6\left(A\cos(t) + B\sin(t)\right) = \cos(t)$$

and by linear independence of cos and sin, we need the coefficients of the cos and sin terms to balance; that is, for the cos terms, we have $-A + 5B + 6A = 1$ and for the sin terms we have $-B - 5A + 6 = 0$. Solving these equations gives $A = B$ and $A = \frac{1}{10}$. So

$$y_p(t) = \frac{1}{10}\left(\cos(t) + \sin(t)\right).$$

(3) Therefore,

$$y(t) = ae^{-3t} + be^{-2t} + \frac{1}{10}\left(\cos(t) + \sin(t)\right)$$

and to find the values of a and b, we use the initial conditions. For $y(0) = 1$, we have

$$1 = a + b + \frac{1}{10}(1 + 0)$$

and for $y'(0) = 0$ we have

$$0 = -3a + -2b + \frac{1}{10}(0 + 1)$$

and solving these two coupled linear systems for a and b we find $a = -\frac{17}{10}$ and $b = \frac{13}{5}$, so

$$\boxed{y(t) = -\frac{17}{10}e^{-3t} + \frac{13}{5}e^{-2t} + \frac{1}{10}\left(\cos(t) + \sin(t)\right).}$$

Problem 2: Numerically solve the IVP in Eq. (1) by using the Runge-Kutta method given by the following Butcher array:

0	0	0	0	0
1/3	1/3	0	0	0
2/3	−1/3	1	0	0
1	1	−1	1	0
	1/8	3/8	3/8	1/8

In particular:

a) Write out the IVP as a system of first-order ODEs.

Solution:

Define $\mathbf{y} \in \mathbb{R}^2$ where $\mathbf{y}_1 = y$ and $\mathbf{y}_2 = y'$. Then

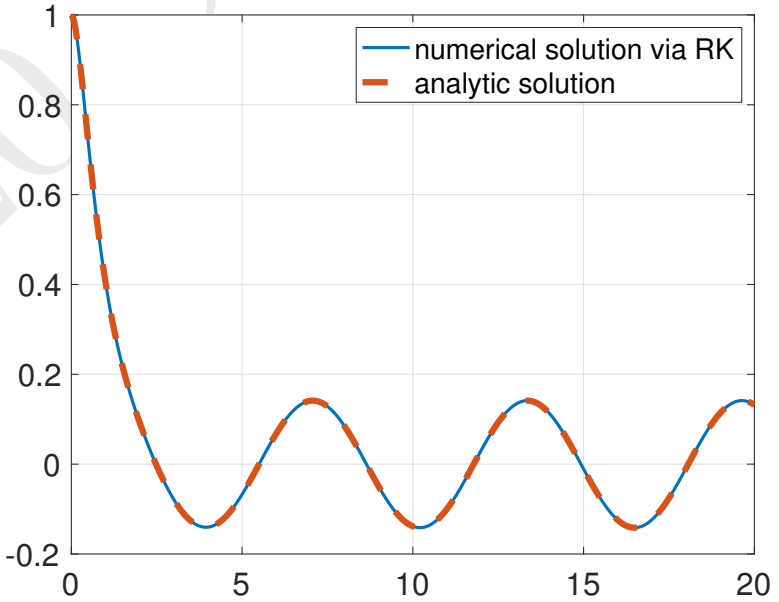
$$\mathbf{y}' = \begin{bmatrix} y' \\ y'' \end{bmatrix} = \begin{bmatrix} \mathbf{y}_2 \\ -5\mathbf{y}_2 - 6\mathbf{y}_1 + \cos(t) \end{bmatrix}$$

There are many notations and ways you could write this out, so it doesn't have to look exactly like the above.

b) Implement the given Runge-Kutta method (for a uniform stepsize h) and, choosing a reasonable value of h, plot your numerical approximation as a function of t for $t \in [0, 20]$. *Hint*: use `ode45` in Matlab or `scipy.integrate.solve_ivp` in Python to make sure that you've implemented the ODE correctly, then code the given Runge-Kutta method and solve that way. *Optional*: also plot the analytic solution you found in problem 1.

Solution:

It should look like this figure (it wasn't necessary to plot the analytic solution, but since we have it, it's nice to plot it to make sure you didn't make any mistakes).



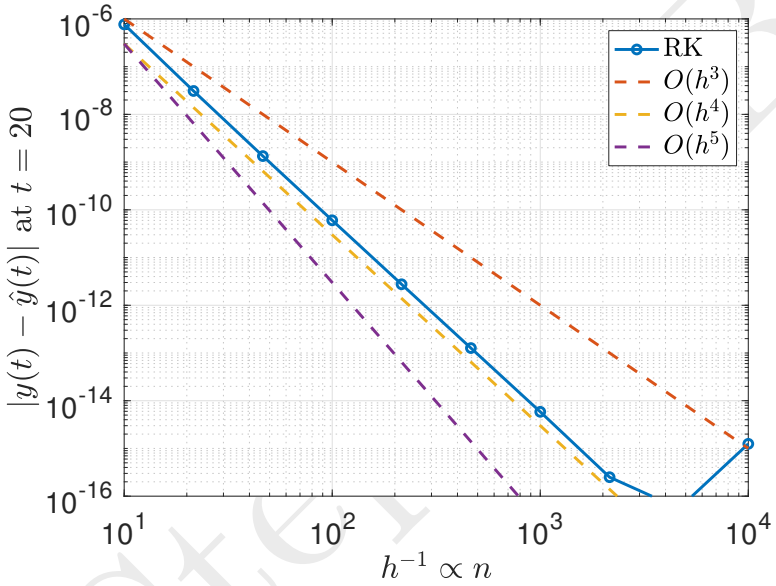
Note: my debugging process for making the code was the following steps: (1) implement the right-hand-side and solve with `ode45` in Matlab or `scipy.integrate.solve_ivp` in

Python, (2) write my own solver that did forward Euler, and then finally (3) I implemented the full RK scheme. If you jump to the full RK scheme right away, it makes it harder to track down bugs (and you might not have faith that your individual components are working).

c) Let $\hat{y}$ represent the numerical solution you found using the Runge-Kutta method, and let y be the true solution you found. Define the error ε to be the absolute value of the difference between your approximation at $t = 20$ and the true solution at $t = 20$. Plot ε as a function of the stepsize h, and use this to determine the order of the error (e.g., $O(h)$ or $O(h^3)$ etc.). Choose the scales (logarithmic or linear) of your plot wisely.

Solution:

It should look like this figure (both axes should be logarithmically scaled to get full credit):



From this we see that it is 4th order, i.e., $O(h^4)$.

d) Include the code you used to solve the IVP (including your plotting code is optional)

Solution:

Matlab code:

```
function dydt = HW9_RK_rhs(t,y)
% y is really a 2D array, with y and y'
dydt = zeros(2,1);
dydt(1) = y(2);  % y' = y'
dydt(2) = -5*y(2) - 6*y(1) + cos(t);
```

and

```
function [t,w] = HW9_RK( f, tspan, y0, h )
if nargin < 4 || isempty(h), h = 0.1; end

a = tspan(1); b = tspan(2);
n   = diff(tspan)/h;
t   = linspace(a,b,ceil(n)+1);
if length(t) < 2, error('choose smaller stepsize'); end
h   = t(2) - t(1);  % might not be exactly the same...
w   = zeros(length(y0),length(t));
w(:,1)= y0;
```

```

for i = 1:length(t)-1
    tt    = t(i);               % for notational convenience
    ww    = w(:,i);             % for notational convenience

    k1    = h*f(tt,       ww);
    k2    = h*f(tt+h/3,   ww+k1/3);
    k3    = h*f(tt+2*h/3, ww-k1/3+k2);
    k4    = h*f(tt+h,     ww+k1-k2+k3);
    w(:,i+1)    = ww + (k1+3*(k2+k3)+k4)/8;
end
```

and the driver to make the plots

```
y0 = [1,0];
T  = 20;
h   = 1e-4;
[tGrid,y] = HW9_RK( @HW9_RK_rhs, [0,T], y0, h );

% Plot for part (b)
plot(tGrid,y(1,:),'-','linewidth',2,'DisplayName','numerical solution via RK
    ');
hold all
set(gca,'fontsize',20);  legend();  grid on
y_p      = @(t) (sin(t)+cos(t))/10;
y_true   = @(t) -17/10*exp(-3*t) + 13/5*exp(-2*t) + y_p(t);
plot(tGrid,y_true(tGrid),'--','linewidth',4,'DisplayName','analytic solution
    ');
export_fig 'HW9_plotRK' -pdf -transparent

% Plot for part (c)
hList   = logspace(-1,-4,10);
errList = [];
for h = hList
    [tGrid,y] = HW9_RK( @HW9_RK_rhs, [0,T], y0, h );
    errList(end+1) = abs( y(1,end) - y_true(T) );
end
figure(2); clf;
loglog(1./hList,errList,'o-', 'linewidth',2,'DisplayName','RK');
hold all
loglog(1./hList,1e-3*hList.^3,'--','linewidth',2,'DisplayName','$O(h^3)$');
loglog(1./hList,3e-3*hList.^4,'--','linewidth',2,'DisplayName','$O(h^4)$');
loglog(1./hList,30e-3*hList.^5,'--','linewidth',2,'DisplayName','$O(h^5)$');
set(gca,'fontsize',20);
xlabel('$h^{-1} \propto n$','interpreter','latex');
ylabel('$|y(t)-\hat y(t)|$ at $t=20$','interpreter','latex');
grid on;  hl=legend(); hl.Interpreter = 'latex';
ylim([1e-16,Inf]);
export_fig 'HW9_errorRK' -pdf -transparent
```